МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6_1

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема. Применение хеш-таблицы для поиска данных в двоичном файле с записями фиксированной длины

Выполнил студент группы ИКБО-42-23 Голев С.С.

Принял ассистент Муравьёва Е.А.

Москва 2024

ОГЛАВЛЕНИЕ

ЦЕЛЬ РАБОТЫ

Получить навыки по разработке хеш-таблиц и их применении при поиске данных в других структурах данных (файлах).

1.УСЛОВИЯ ЗАДАЧ

Задание 1:

Устная часть.

Задание 2:

Разработайте приложение, которое использует хеш-таблицу (пары «ключ – хеш») для организации прямого доступа к элементам динамического множества полезных данных (записи в файле).

1. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте в таблице

2. Метод разрешения коллизии также представлен в индивидуальном варианте.

Задание 3:

Управление бинарным файлом посредством хеш-таблицы.

В заголовочный файл подключить заголовочные файлы: управления хеш-таблицей, управления двоичным файлом.

Открытый адрес (двойное хеширование)	Книга: ISBN – двенадцатизначное число, Автор, Название.

2. ОТЧЁТ ПО ЗАДАНИЮ 2

2.1. Определение структуры

АТД HashTable
{
Данные.
table – указатель на блок с данными хэщ-таблицы.
{
Данные.
key – ключ записи.
author – хэш.
name – хэш.
Операции.
1)Вывод содержимого блока данных.
//Предусловие. Блок хэш-таблицы.
//Постусловие. Содержимое блока.
printBlock();
}
size – количество элементов в хэш-таблице.
Операции.
1) Создание хэш-таблицы;
//Предусловие. Количество элементов
//Постусловие. Пустая хэш-таблица с выделенными местами
createTable(int size);
2) Первая хэш-функция;
//Предусловие. Ключ и размер таблицы,
//Постусловие. Значение хэш-функции
hashFunc1(unsigned long long key, int size);
3) Вторая хэш-функция;
//Предусловие. Ключ и размер таблицы,
//Постусловие. Значение хэш-функции
hashFunc2(unsigned long long key, int size);
4) Вставка элемента в таблицу;
//Предусловие. Ключ и хэш,
//Постусловие. Таблица с новым элементом
insertInHashTable(unsigned long long key, char* author, char* name);
5) Поиск по ключу;
//Предусловие. Ключ,
//Постусловие. Хэш соответствующий ключу
findKey(unsigned long long key);
6) Удаление по ключу;
//Предусловие. Ключ,
//Постусловие. Таблица без заданного элемента

```
        deleteKey(unsigned long long key);
        7) Рехеширование;
        //Предусловие. Таблица,
        //Постусловие. Новая таблица, большая в размерах
        reHashTable();
        8) Вывод содержимого таблицы;
        //Предусловие. Таблица,
        //Постусловие. Вывод таблицы на экран
        printHashTable();
        9) Коэффициент нагрузки;
        //Предусловие. Таблица,
        //Постусловие. Коэффициент нагрузки этой таблицы
        coefTable();
}
```

2.2. Код используемый в программе

Опишем функции, используемые в программе для решения задач.

```
void createTable(int size)
   {
      this->size = size;
      this->table = new HashBlock[size];
   } }
```

Рисунок 2.1 – Функция конструктор

```
unsigned long long hashFunc1(unsigned long long key, int size)
{
return key % size;
}
 unsigned long long hashFunc2(unsigned long long key, int size)
{
return  (1 + key % (size + 2));
}
```

Рисунок 2.2 – Хэш-функциии

```
void insertInHashTable(unsigned long long key, char* author, char* name)
   {
      if (this -> coefTable() > 0.7) { reHashTable(); }
      unsigned long long index = hashFunc1(key, this->size);
      unsigned long long index_1 = index;
      unsigned long long index_2 = hashFunc2(key, this->size);

      int i = 0;
      while (table[index].flag)
      {
         index = index_1 + i * index_2;
         i++;
         if (index >= size) { reHashTable(); }
      }
      table[index].key = key;
      table[index].author = author;
      table[index].name = name;
      table[index].flag = true;
   }
```

Рисунок 2.3 – Функция вставки

```
HashBlock findKey(unsigned long long key)
   {
      unsigned long long index = hashFunc1(key, this->size);
      unsigned long long index_1 = index;
      unsigned long long index_2 = hashFunc2(key, this->size);
      int i = 0;
      while (table[index].flag)
      {
         if (table[index].key == key)
         {
            return table[index];
         }
         index = index_1 + i * index_2;
         i++;
      }
      std::cout << "Key not found" << std::endl;
      return HashBlock();
   }
```

Рисунок 2.4 – Функция поиска

```
void deleteKey(unsigned long long key) {
      unsigned long long index = hashFunc1(key, this->size);
      unsigned long long index_1 = index;
      unsigned long long index_2 = hashFunc2(key, this->size);
      int i = 0;
      while (table[index].flag)
      {
         if (table[index].key == key)
         {
            table[index].flag = false;
            return;
         }
         index = index_1 + i * index_2;
         i++;
      }
      std::cout << "Key not found" << std::endl;
   }
```

Рисунок 2.5 – Функция удаления

```
void reHashTable() {
	int new_size = size * 2;
	HashBlock* newTable = new HashBlock[new_size];

	for (int i = 0; i < size; i++)
	{
	   if (table[i].flag)
	   {

	      unsigned long long newIndex = hashFunc1(table[i].key, new_size);
	      unsigned long long newIndex_1 = newIndex;
	      unsigned long long newIndex_2 = hashFunc2(table[i].key, new_size);
	      int i = 0;

	      while (newTable[newIndex_1].flag)
	      {
	         newIndex = newIndex_1 + i * newIndex_2;
	         i++;
	      }
	      newTable[newIndex].key = table[i].key;
	      newTable[newIndex].author = table[i].author;
	      newTable[newIndex].name = table[i].name;
	      newTable[newIndex].flag = true;
	   }
	}
	delete[] table;
	this->table = newTable;
	this->size = new_size;
   }
```

Рисунок 2.6 – Функция рехэширования

```
void printHashTable() {
	int num = 1;
	for (int i = 0; i < size; i++) {
	   if (table[i].flag) {
	      std::cout << num << '\t';
	      table[i].printBlock();
	      num++;
	   }
	   else {
	      std::cout << num << '\t';
	      std::cout << "--------------------------------" << std::endl;
	      num++;
	   }
	}
   }
```

Рисунок 2.7 – Функция вывода

```
float coefTable() {
        int num = 0;
        for (int i = 0; i < size; i++)
        {
            if (table[i].flag)
            {
                num++;
            }
        }
        return num/size;
    }
```

Рисунок 2.8 – Функция коэффициента нагрузки

3. ОТЧЁТ ПО ЗАДАНИЮ 3

3.1. Код используемый в программе

Опишем функции, используемые в программе для решения задач.

```
void convertTextToBinary(const char* textFileName, const char* binaryFileName) {
    std::ifstream inputFile(textFileName);
    std::ofstream outputFile(binaryFileName, std::ios::out | std::ios::binary);

    if (!inputFile.is_open())
    {
        std::cerr << "Ошибка открытия текстового файла" << std::endl;
        return;
    }
    if (!outputFile.is_open())
    {
        std::cerr << "Ошибка создания двоичного файла" << std::endl;
        return;
    }

    s_book record;
    while (inputFile >> record.ISBN >> record.author >> record.name)
    {
        outputFile.write((char*)(&record), sizeof(s_book));
    }

    inputFile.close();
    outputFile.close();
}
```

Рисунок 3.1 – Перевод текстового файла в бинарный

```
void deleteKeyBin(const char* binaryFileName, unsigned long long delKey)
{
   std::fstream bin(binaryFileName, std::ios::binary | std::ios::in);
   std::fstream temp("temp_file.dat", std::ios::binary | std::ios::out);

   if (!bin.is_open())
   {
      std::cerr << "Ошибка открытия файла\n";
      return;
   }

   if (!temp.is_open())
   {
      std::cerr << "Ошибка открытия временного файла\n";
      bin.close();
      return;
   }

   std::string s = intToString(delKey);
   const char* delChar = s.c_str();
   s_book record;

   while (bin.read((char*)&record, sizeof(s_book)))
   {
      int result = strcmp(record.ISBN, delChar);
      if (result != 0)
      {
         temp.write((char*)&record, sizeof(s_book));
      }
   }
   bin.close();
   temp.close();

   remove(binaryFileName);
   rename("temp_file.dat", binaryFileName);
}
```

Рисунок 3.2 – Функция удаления по ключу в бинарном файле

```
s_book findKeyBin(const char* binaryFileName, unsigned long long delKey)
{
    std::ifstream bin(binaryFileName, std::ios::binary);
    if (!bin)
    {
        std::cerr << "Не удалось открыть файл для записи";
        return s_book();
    }

    std::string s = intToString(delKey);
    const char* delChar = s.c_str();
    s_book record;

    while (bin.read((char*)&record, sizeof(s_book)))
    {
        int result = strcmp(record.ISBN, delChar);
        if (result == 0)
        {
            bin.close();
            return record;
        }
    }

    bin.close();
    return record;
}
```

Рисунок 3.3 – Функция поиска по ключу в бинарном файле

```
void printAllRecords(const char* binaryFileName) {
    std::ifstream inputFile(binaryFileName, std::ios::binary);
    if (!inputFile.is_open())
    {
        std::cout << "Ошибка открытия двоичного файла" << std::endl;
        return;
    }

    s_book record;
    int num = 1;
    while (inputFile.read(reinterpret_cast<char*>(&record), sizeof(s_book)))
    {
        std::cout <<num <<'\t' << "ISBN: " << record.ISBN << ",\tAuthor: " << record.author <<
",\tName: " << record.name << std::endl;
        num++;
    }

    inputFile.close();
}
```

Рисунок 3.4 – Функция вывода бинарного файла

```
HashTable tabFromTextFile(const char* textInputFile)
{
   std::vector<s_book> vec;
   s_book record;
   int i = 0;

   std::ifstream inputFile(textInputFile);
   while (inputFile >> record.ISBN >> record.author >> record.name)
   {
      vec.push_back(record);
      i++;
   }
   inputFile.close();

   HashTable table;
   table.createTable(i);
   for (int in = 0; in < i; in++)
      table.insertInHashTable(stringToInt(vec[in].ISBN), vec[in].author, vec[in].name);

   return table;
}
```

Рисунок 3.5 – Функция перевода текстового файла в таблицу

```
void fromFileToHashTab(HashTable& hashtab, unsigned long long keyBin, const char*
bin_name)
{
   s_book record = findKeyBin(bin_name, keyBin);
   hashtab.insertInHashTable(keyBin, record.author,record.name);
}

void deleteFromTabnFile(HashTable hashtab, unsigned long long key, const char*
binaryFileName)
{
   hashtab.deleteKey(key);
   deleteKeyBin(binaryFileName, key);
}
void findInTabnFile(HashTable hashtab, unsigned long long key, const char* binaryFileName)
{
   std::cout << "Хэш-таблица:\t";
   hashtab.findKey(key).printBlock();
   std::cout << "Бинарный файл:\t";
   findKeyBin(binaryFileName, key).printBlock();
}
```

Рисунок 3.5 – Функции для работы и с бинарным файлом и таблицей

3.2. Результаты тестирования

Протестируем заданные функции на файле из 10 элементов.

```
1. Вывод содержимого бинарного файла.
2. Вывод содержимого хэш-таблицы.
3. Поиск по ключу.
4. Удаление по ключу из таблицы и файла.
5. Вставка по ключу из файла в таблицу.
6. Завершение.
1

№       |       Ключ            |       Хэш
1       ISBN: 111111111100,     Author: Daniel, Name: one
2       ISBN: 111111111101,     Author: Alice,  Name: two
3       ISBN: 111111111102,     Author: Emily,  Name: three
4       ISBN: 111111111103,     Author: Chloe,  Name: four
5       ISBN: 111111111104,     Author: Max,    Name: five
6       ISBN: 111111111105,     Author: Daniel, Name: onec
7       ISBN: 111111111106,     Author: Alice,  Name: twoc
8       ISBN: 111111111107,     Author: Emily,  Name: threec
9       ISBN: 111111111108,     Author: Chloe,  Name: fourc
10      ISBN: 111111111109,     Author: Max,    Name: fivec
```

Рисунок 3.6 – Тестирование функции, вывода бинарного файла

```
1. Вывод содержимого бинарного файла.
2. Вывод содержимого хэш-таблицы.
3. Поиск по ключу.
4. Удаление по ключу из таблицы и файла.
5. Вставка по ключу из файла в таблицу.
6. Завершение.
2

№       |       Ключ            |       Хэш
1       ISBN: 111111111100      Author: Daniel  Name: one
2       ISBN: 111111111101      Author: Alice   Name: two
3       ISBN: 111111111102      Author: Emily   Name: three
4       ISBN: 111111111103      Author: Chloe   Name: four
5       ISBN: 111111111104      Author: Max     Name: five
6       ISBN: 111111111105      Author: Daniel  Name: onec
7       ISBN: 111111111106      Author: Alice   Name: twoc
8       ISBN: 111111111107      Author: Emily   Name: threec
9       ISBN: 111111111108      Author: Chloe   Name: fourc
10      ISBN: 111111111109      Author: Max     Name: fivec
```

Рисунок 3.7 – Тестирование функции, вывода хэш-таблицы

Протестируем функции взаимодействия с таблицей и файлом.

```
1. Вывод содержимого бинарного файла.
2. Вывод содержимого хэш-таблицы.
3. Поиск по ключу.
4. Удаление по ключу из таблицы и файла.
5. Вставка по ключу из файла в таблицу.
6. Завершение.
3
Введите ключ (12-значное число): 111111111103
Хэш-таблица:    ISBN: 111111111103      Author: Chloe   Name: four
Бинарный файл:  ISBN: 111111111103      Author: Chloe   Name: four
```

Рисунок 3.8 – Тестирование функции, вывода элемента

```
1. Вывод содержимого бинарного файла.
2. Вывод содержимого хэш-таблицы.
3. Поиск по ключу.
4. Удаление по ключу из таблицы и файла.
5. Вставка по ключу из файла в таблицу.
6. Завершение.
2

№       |       Ключ            |       Хэш
1       ISBN: 111111111100      Author: Daniel  Name: one
2       ISBN: 111111111101      Author: Alice   Name: two
3       ISBN: 111111111102      Author: Emily   Name: three
4       -------------------------------
5       ISBN: 111111111104      Author: Max     Name: five
6       ISBN: 111111111105      Author: Daniel  Name: onec
7       ISBN: 111111111106      Author: Alice   Name: twoc
8       ISBN: 111111111107      Author: Emily   Name: threec
9       ISBN: 111111111108      Author: Chloe   Name: fourc
10      ISBN: 111111111109      Author: Max     Name: fivec
```

Рисунок 3.9 – Тестирование функции, удаления элемента

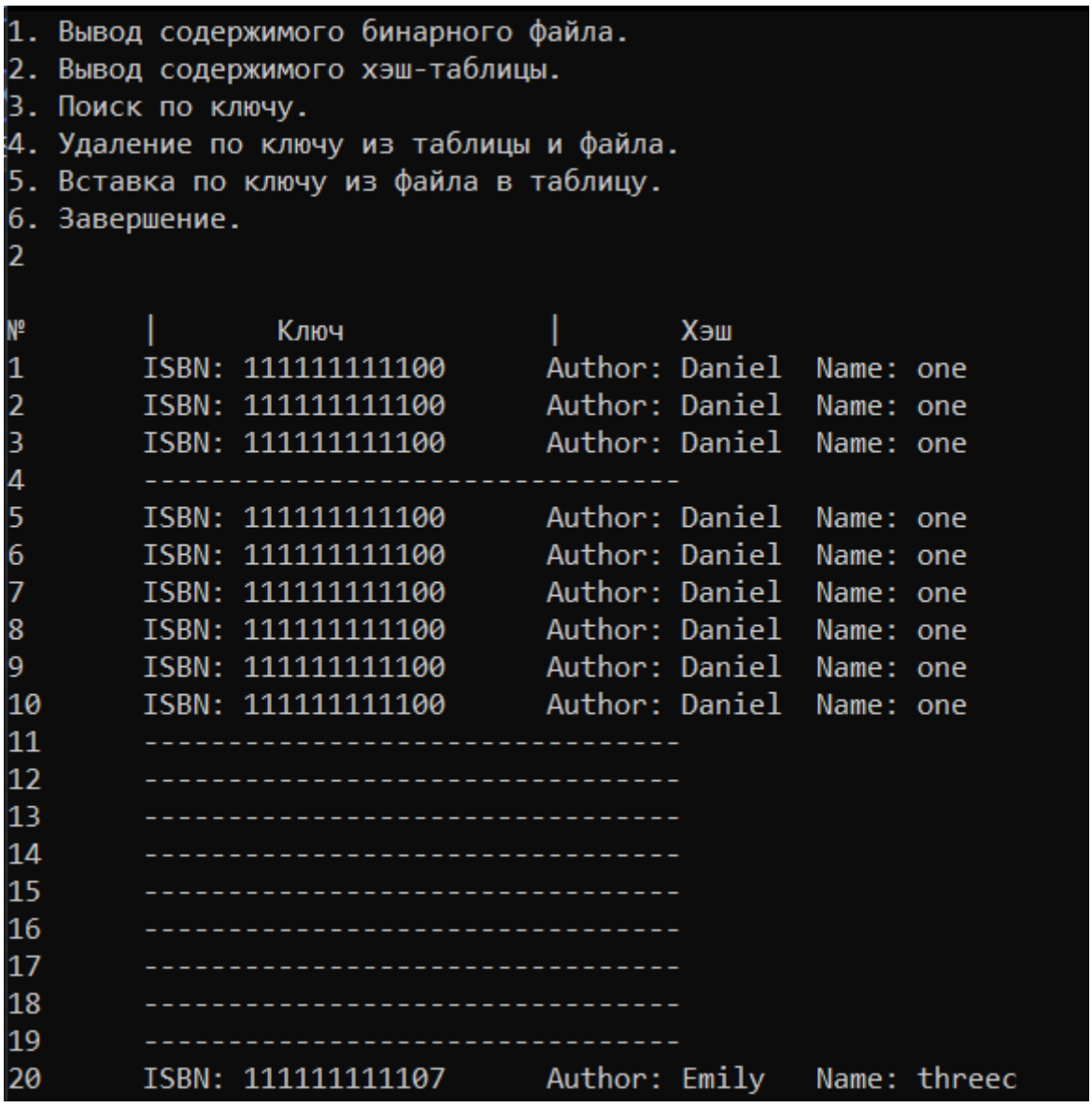

Рисунок 3.10 – Тестирование функции, вставки элемента

Так как новый элемент не помещается в предыдущую таблицу, происходит рехеширование.

4.ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Лекции по Структуры и алгоритмы обработки данных / Рысин М. Л. Москва, МИРЭА — Российский технологический университет.

2. Материалы по дисциплине Структуры и алгоритмы обработки данных / Скворцова Л. А. Москва, МИРЭА — Российский технологический университет.